EECS 3201 Digital Logic Laboratory

Lab4: Shot Clock

Objective

In this lab you will use sequential logic design to create a shot clock appropriate for basketball.

Background

In most basketball leagues, a team with possession of the ball has a given number of seconds to take a shot on the basket, or else they lose possession of the ball. In the NBA, WNBA, and OUA, the shot clock is set to 24 seconds, while in the NCAA, the shot clock is set to 30 seconds.

Shot Clock

Your task is to create a shot clock with the following features, starting with a 24-second clock:

- The clock counts down from 24 to zero, displaying the time remaining in seconds (as a decimal integer) on the seven-segment displays;
- If the clock reaches zero, the display holds at a zero until reset; to signal the time expiry the rightmost decimal point should flash on and off every 0.5 s when the count is 00 (hint you can AND or OR with the clock to flash at 0.5s); and
- At any time, whether it has reached zero or not, the clock can be reset by pressing a button; once reset, it immediately starts counting down again from the maximum time.

Your countdown should be in decimal, if you can only implement in hexadecimal the circuit can be demonstrated and submitted for a small penalty.

Additional features:

- The clock can be paused by pressing a (second) button, and un-paused by pressing the same button.
- The maximum time can be selected to either 24 or 30 seconds, via a switch.

Full marks will require correct implementation of all features and a decimal display.

Verilog code for a clock divider (i.e., that takes the 50 MHz system clock as input, and gives a 1 Hz clock as output) is provided below. You may use this code as-is, modify it, or disregard it, as you like.

```
module ClockDivider(cin,cout);

   // Based on (corrected) code from fpga4student.com
   // cin is the input clock; if from the DE10-Lite,
   // the input clock will be at 50 MHz
   // The clock divider toggles cout every 25 million
   // cycles of the input clock

   input cin;
   output reg cout;

   reg[31:0] count;
   parameter D = 32'd25000000;

   always @(posedge cin)
   begin
      count <= count + 32'd1;
         if (count >= (D-1)) begin
            cout <= ~cout;
            count <= 32'd0;
         end
   end

endmodule
```

You may use your code from previous labs to complete this lab. If you do, make a note of this in a comment in your code (e.g. // this file previously submitted as part of Lab 3).

The implemented circuits must be demonstrated to the TA who will note a completed lab and ask questions about your design. When implementing the circuit be sure to use the switches and lights as described to make it easy to demonstrate your circuits.

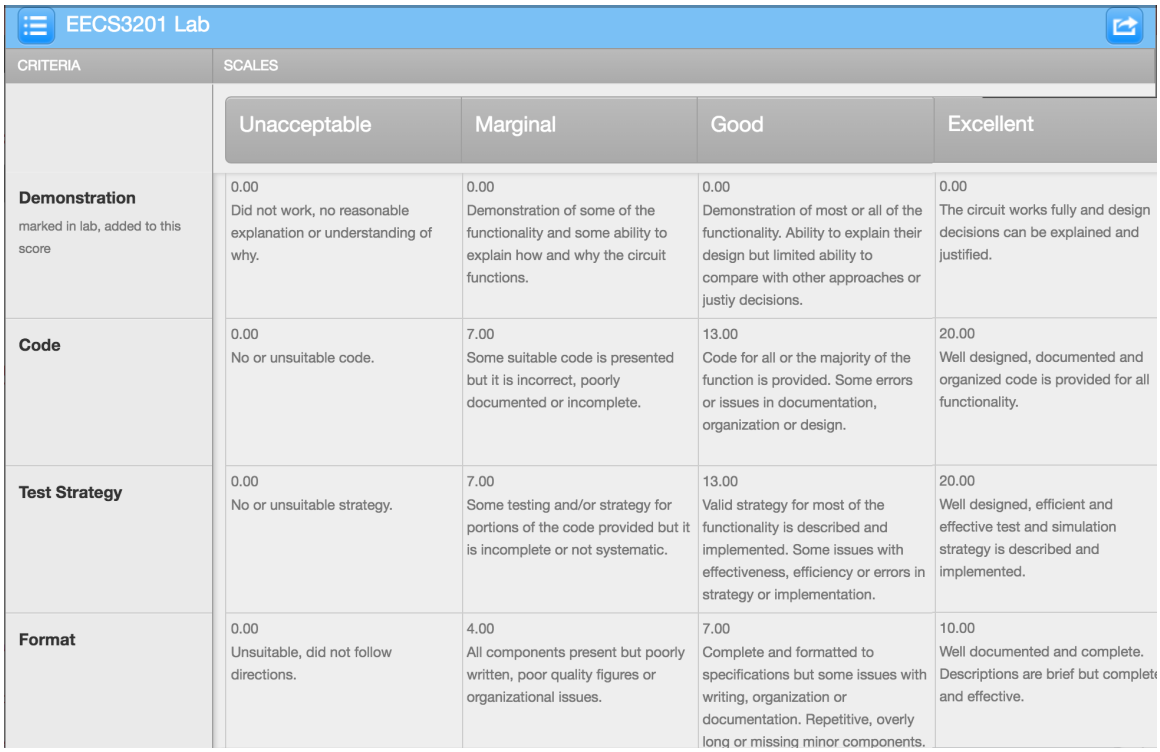
Evaluation

Lab demonstration, in-lab explanations and answers, debug and test approach. [2 marks for fully working]

Lab report containing [graded out of 50, weight 3 marks of the overall lab grade]:

- Explanation of design approach and design documentation such as truth tables, maps, Boolean expressions, timing diagram, state tables, and other design aids.
- Fully documented Verilog source including top level files (submit as .v files)

- Test patterns and/or a testing strategy



EECS3201 Lab

CRITERIA	SCALES			
	Unacceptable	Marginal	Good	Excellent
Demonstration marked in lab, added to this score	0.00 Did not work, no reasonable explanation or understanding of why.	0.00 Demonstration of some of the functionality and some ability to explain how and why the circuit functions.	0.00 Demonstration of most or all of the functionality. Ability to explain their design but limited ability to compare with other approaches or justiy decisions.	0.00 The circuit works fully and design decisions can be explained and justified.
Code	0.00 No or unsuitable code.	7.00 Some suitable code is presented but it is incorrect, poorly documented or incomplete.	13.00 Code for all or the majority of the function is provided. Some errors or issues in documentation, organization or design.	20.00 Well designed, documented and organized code is provided for all functionality.
Test Strategy	0.00 No or unsuitable strategy.	7.00 Some testing and/or strategy for portions of the code provided but it is incomplete or not systematic.	13.00 Valid strategy for most of the functionality is described and implemented. Some issues with effectiveness, efficiency or errors in strategy or implementation.	20.00 Well designed, efficient and effective test and simulation strategy is described and implemented.
Format	0.00 Unsuitable, did not follow directions.	4.00 All components present but poorly written, poor quality figures or organizational issues.	7.00 Complete and formatted to specifications but some issues with writing, organization or documentation. Repetitive, overly long or missing minor components.	10.00 Well documented and complete. Descriptions are brief but complete and effective.